**Experiment 8: Understanding Clustering - I**

**Objective**:

* Develop an understanding of how to perform k-means on a data set.
* Develop an understanding of the use of objective function to select the best possible value of k in k-means clustering.
* Learn how to implement KMeans with PCA

**Time Required**: 3 hrs

**Programming Language**: Python

**Software Required**: Anaconda

**Introduction**

The technique to segregate datasets into various groups, on basis of having similar features and characteristics, is being called Clustering. The groups being formed are being known as Clusters. Clustering technique is used as a data analysis technique for discovering interesting patterns in data, such as groups of customers based on their behavior and in various Field such as Image recognition, Spam Filtering. Clustering is being used in Unsupervised Learning Algorithm in Machine Learning as it can be segregated multivariate data into various groups, without any supervisor, on basis of common pattern hidden inside the datasets.

There are many clustering algorithms to choose from and no single best clustering algorithm for all cases. Instead, it is a good idea to explore a range of clustering algorithms and different configurations for each algorithm. In this lab we will be learning and understanding the implementation of k-means clustering algorithm.

**KMeans Clustering:** Kmeans Algorithm is an Iterative algorithm that divides a group of n datasets into k subgroups or clusters based on the similarity and their mean distance from the centroid of that particular subgroup formed.

K, here is the pre-defined number of clusters to be formed by the Algorithm. If K=3, It means the number of clusters to be formed from the dataset is 3

**Task 1:**

You have to solve the customer segmentation problem by using KMeans clustering and the dataset “Mall\_Customers.csv”.

***Steps to follow:***

1. Import the important libraries including the following

from sklearn import preprocessing

1. Load and view the dataset
2. Apply feature scaling using MinMaxScaler. You can use the following code:

#Feature Scaling

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scale = scaler.fit\_transform(df[['Annual Income (k$)','Spending Score (1-100)']])

df\_scale = pd.DataFrame(scale, columns = ['Annual Income (k$)','Spending Score (1-100)']);

df\_scale.head(5)

1. Apply KMeans with 2 clusters

#Applying KMeans

from sklearn.cluster import KMeans

import sklearn.cluster as cluster

km=KMeans(n\_clusters=2)

y\_predicted = km.fit\_predict(df[['Annual Income (k$)','Spending Score (1-100)']])

y\_predicted

1. Find the centroid of the two clusters by using the attribute ‘cluster\_centers\_’ as shown below:

#Find the centroid

km.cluster\_centers\_

1. Visualize the results by using the scatterplot from seaborn library

#Visualize Results

df['Clusters'] = km.labels\_

sns.scatterplot(x="Spending Score (1-100)", y="Annual Income (k$)",hue = 'Clusters', data=df,palette='viridis')

**Finding Optimum number of Clusters in K Means**

The tricky part with K-Means clustering is one does not know in advance that in how many clusters the given data can be divided. There are two methods that can be used to find the optimal value of K other than hit and trial but in this lab, we would be using only one which is WCSS.

**Elbow Method with Within-Cluster-Sum of Squared Error (WCSS)**

The Elbow Method is a popular technique for determining the optimal number of clusters. Here, we calculate the Within-Cluster-Sum of Squared Errors (WCSS) for various values of k and choose the k for which WSS first starts to diminish. In the plot of WSS-versus-k, this can be observed as an elbow.

* The Squared Error for a data point is the square of the distance of a point from its cluster center.
* The WSS score is the summation of Squared Errors for all given data points.
* Distance metrics like Euclidean Distance or the Manhattan Distance can be used.

**Task 2:**

Continuing with our task 1,

1. Calculate the WCSS for K=2 to k=12 and calculate the WCSS in each iteration by using the following code:

#Finding optimum value of K

K=range(2,12)

wss = []

for k in K:

kmeans=cluster.KMeans(n\_clusters=k)

kmeans=kmeans.fit(df\_scale)

wss\_iter = kmeans.inertia\_

wss.append(wss\_iter)

1. Plot the WCSS vs K cluster graph

#Plotting the graph

plt.xlabel('K')

plt.ylabel('Within-Cluster-Sum of Squared Errors (WSS)')

plt.plot(K,wss)

***Note:*** You will observe an elbow bend at point 5. It is the point after which WCSS does not diminish much with the increase in value of K.

1. After finding out the optimum value of K, apply KMeans with this value and plot the graph**.**

#Applying KMeans with optimal value of K

kmeans = cluster.KMeans(n\_clusters=5 ,init="k-means++")

kmeans = kmeans.fit(df[['Annual Income (k$)','Spending Score (1-100)']])

df['Clusters'] = kmeans.labels\_

sns.scatterplot(x="Spending Score (1-100)", y="Annual Income (k$)",hue = 'Clusters', data=df,palette='viridis')

**Task 3:**

Apply KMeans clustering after reducing the dimensionality of dataset into two components. You can use the following code for applying PCA:

#Applying PCA

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(df\_scale)

pca\_df = pd.DataFrame(data = principalComponents

, columns = ['principal component 1', 'principal component 2'])

pca\_df.head()